

CS195: Computer Vision

Fine-tuning popular CNNs for image recognition

Wednesday, September 25th, 2024



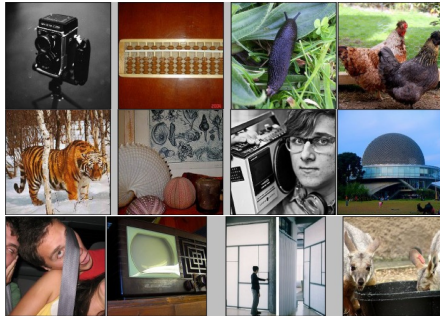
Today's Agenda

- Training vs. Fine-tuning
- Fine-tuning a popular CNN (eg, AlexNet) using an arbitrary dataset

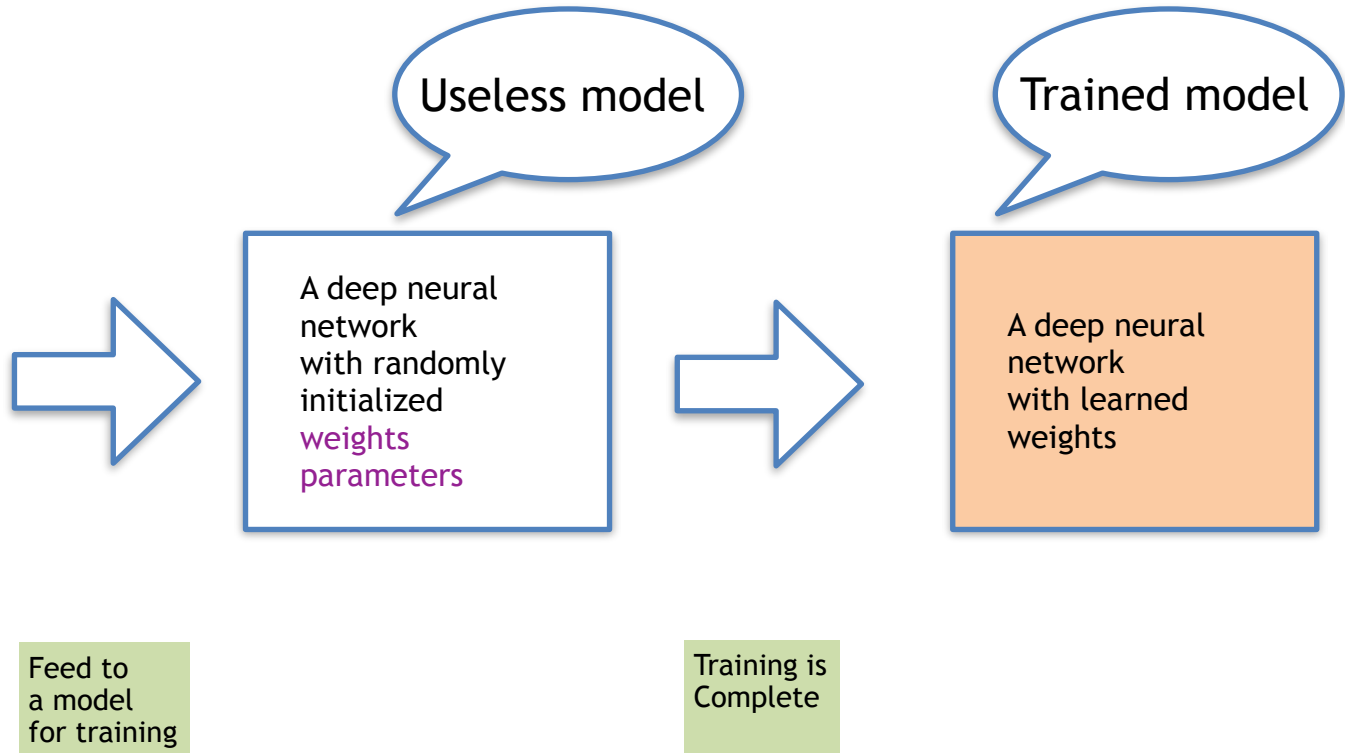
Training a Model

- **Training** refers to the process of training a model from scratch, often on a large and general dataset (e.g., ImageNet for image classification).

IMAGENET

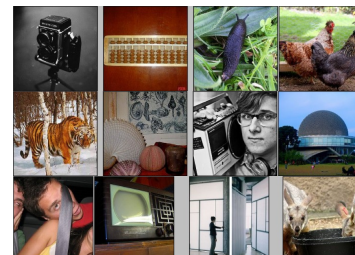


A dataset with over 1 million images



How to train your CNN?

IMAGENET



A dataset with over 1 million images

- If we consider a collection of training examples and Sum the error term over all examples

$$E(\mathbf{w}) = \sum_i Error(\mathbf{x}_i, y_i, \mathbf{w}) = \sum_i (y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

- Minimize errors using an optimization algorithm:
 - Gradient Descent (GD)
 - Stochastic Gradient Descent (SGD)

$E(\mathbf{w})$ term is also known as *loss function*

How to train your CNN?

- Every neural network consists of numerous **weight parameters** across its various layers, which must be learned through an optimization technique.
- There are several optimization techniques:
 - Gradient Descent (GD)
 - Stochastic Gradient Descent (SGD)
 - Adam
 - RMSProp

How to train your CNN?

- Initialize the weight vector at a random position \mathbf{w}^{old} (random set of values)
- Keep doing the following two steps sequentially until the loss function gets to a low value (eg, below a threshold)
 - **Step 1:** calculate how much the loss function would change if we make a small change with respect to one weight component without perturbing any other weight terms. This is the gradient term corresponding to that particular weight. When you put them all together, they become the **gradient vector**:

$$\nabla E(\mathbf{w})$$

- **Step 2:** adjust (or update) the values of the weights based on the **gradient vector** computed in the previous step:

$$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - \eta \nabla E(\mathbf{w})$$

a fixed learning rate

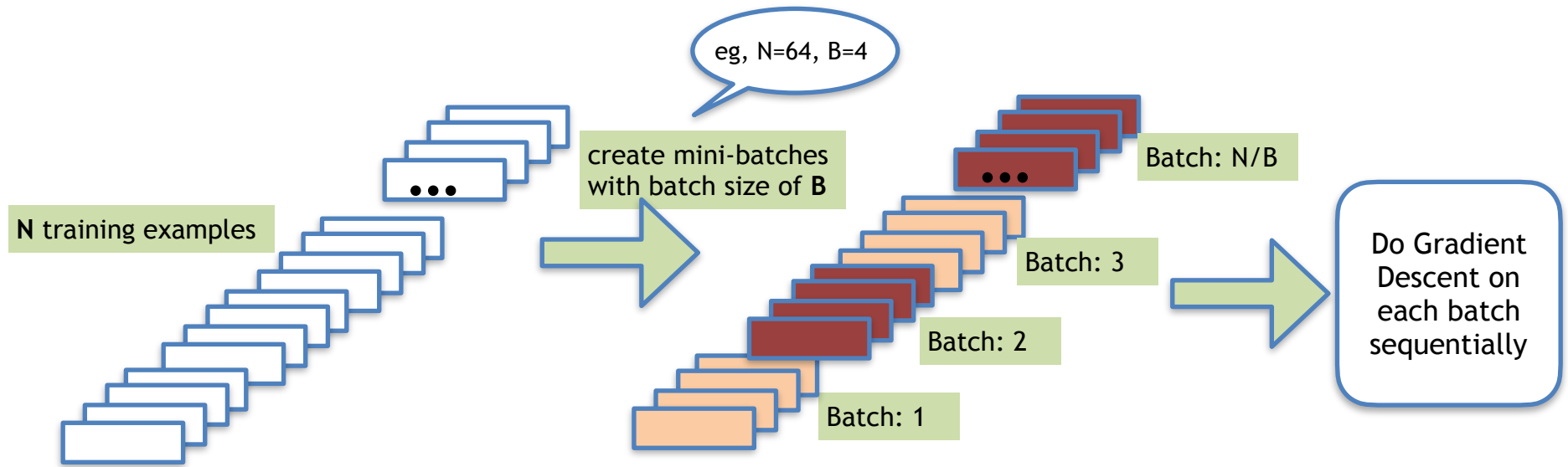
How to train your CNN?

Stochastic Gradient Descent (SGD)

- Keep doing the **Gradient Descent**, but instead of using all the training samples, *use small subset of training samples* picked randomly when computing the **gradient vector**
- divide the entire training data into **mini batches**
- calculate the **gradient vector** based on that batch $\nabla E(\mathbf{w})$
- adjust (or update) the values of the weights based on the **gradient vector** to that batch

$$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - \eta \nabla E(\mathbf{w})$$

Stochastic Gradient Descent (SGD)

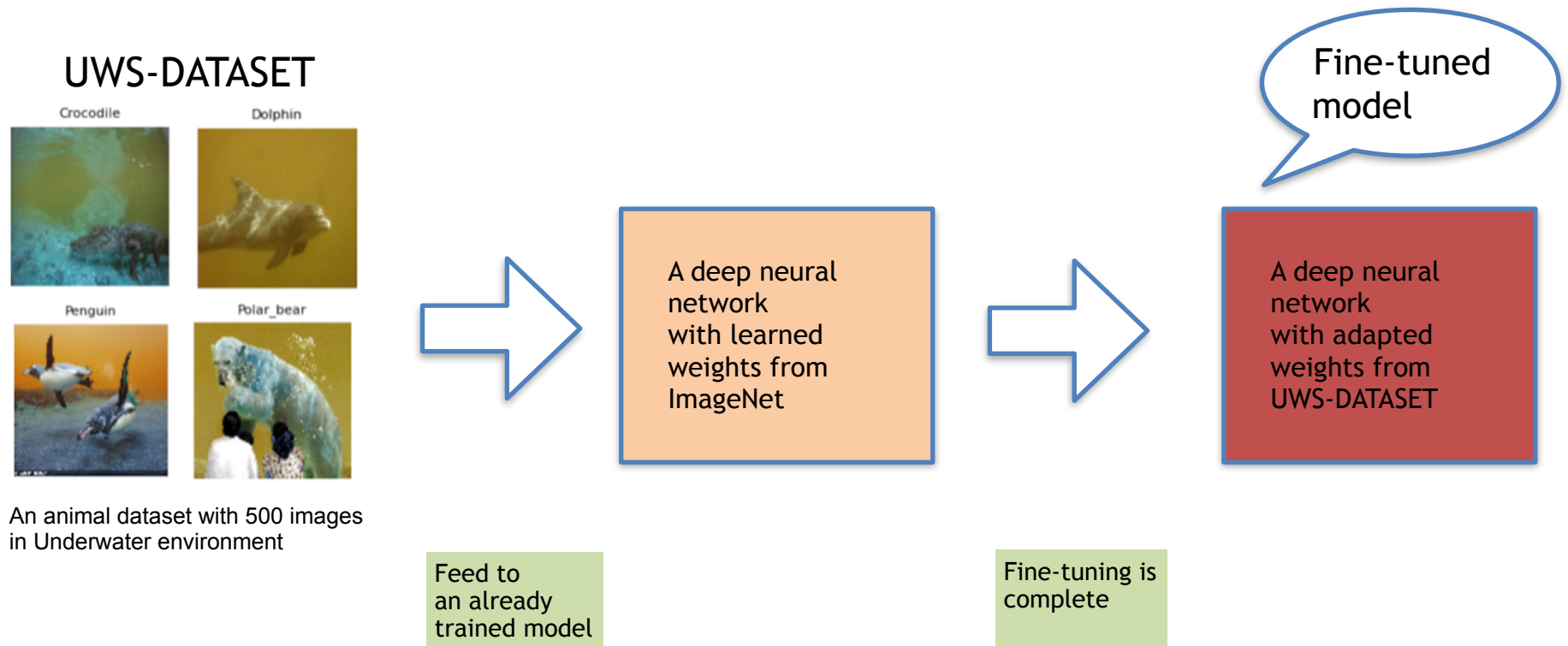


Useful Online Resources for Gradient Descent

- Another [mathematical derivation for Gradient Descent](#)
- One more [mathematical derivation for Gradient Descent](#)
- [Google course's Gradient Descent](#)
- [Visualization of Gradient Descent](#)
- [Visual explanation of Gradient Descent and other optimizers](#)

Fine-tuning a Model

- **Fine-tuning** refers to the process of taking a pre-trained model and further training it on a new or specific dataset. The initial model is often trained on a large and general dataset, e.g., ImageNet, and fine-tuning adapts the model to perform well on a more specific task or dataset.



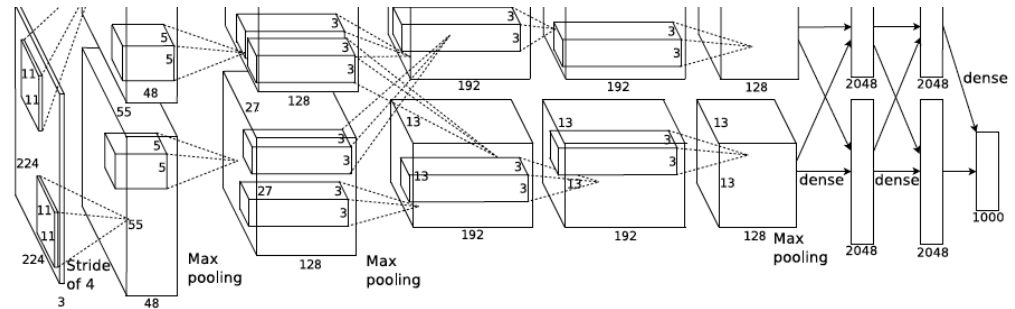
Today's Agenda

- Training vs. Fine-tuning
- Fine-tuning a popular CNN (eg, AlexNet) using an arbitrary dataset

Fine-tuning AlexNet on an Arbitrary Dataset

- Let's use one of the popular CNNs

- LeNet
- AlexNet**
- VGG
- ResNet



- Let's fine-tune AlexNet with a new dataset eg, UWS-DATASET

UWS-DATASET



An animal dataset with 500 images in Underwater environment

Existing Dataset in PyTorch

- Notice these are some of the datasets provided by PyTorch.




Image classification	
<code>Caltech101(root[, target_type, transform, ...])</code>	Caltech 101 Dataset.
<code>Caltech256(root[, transform, ...])</code>	Caltech 256 Dataset.
<code>CelebA(root[, split, target_type, ...])</code>	Large-scale CelebFaces Attributes (CelebA) Dataset
<code>CIFAR10(root[, train, transform, ...])</code>	CIFAR10 Dataset.

Fine-tuning AlexNet on an Arbitrary Dataset

- Download the following dataset and put it into your Google Drive
 - [Underwater Animal Dataset \(partial\)](#)
 - Each image size: **HxWx3**
 - Note that these are color images
 - Each image is associated with a label from **10 classes**
 - Training set of **241** examples and test set of **60** examples

https://analytics.drake.edu/~reza/teaching/cs195_fall24/datasets/uws_v1_partial.zip

Fine-tuning AlexNet on an Arbitrary Dataset

- This is a random dataset of images, unlike the datasets provided by PyTorch.

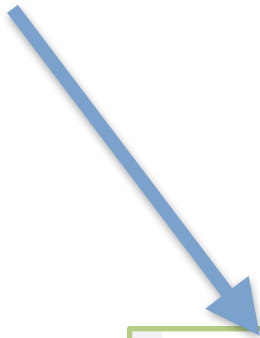
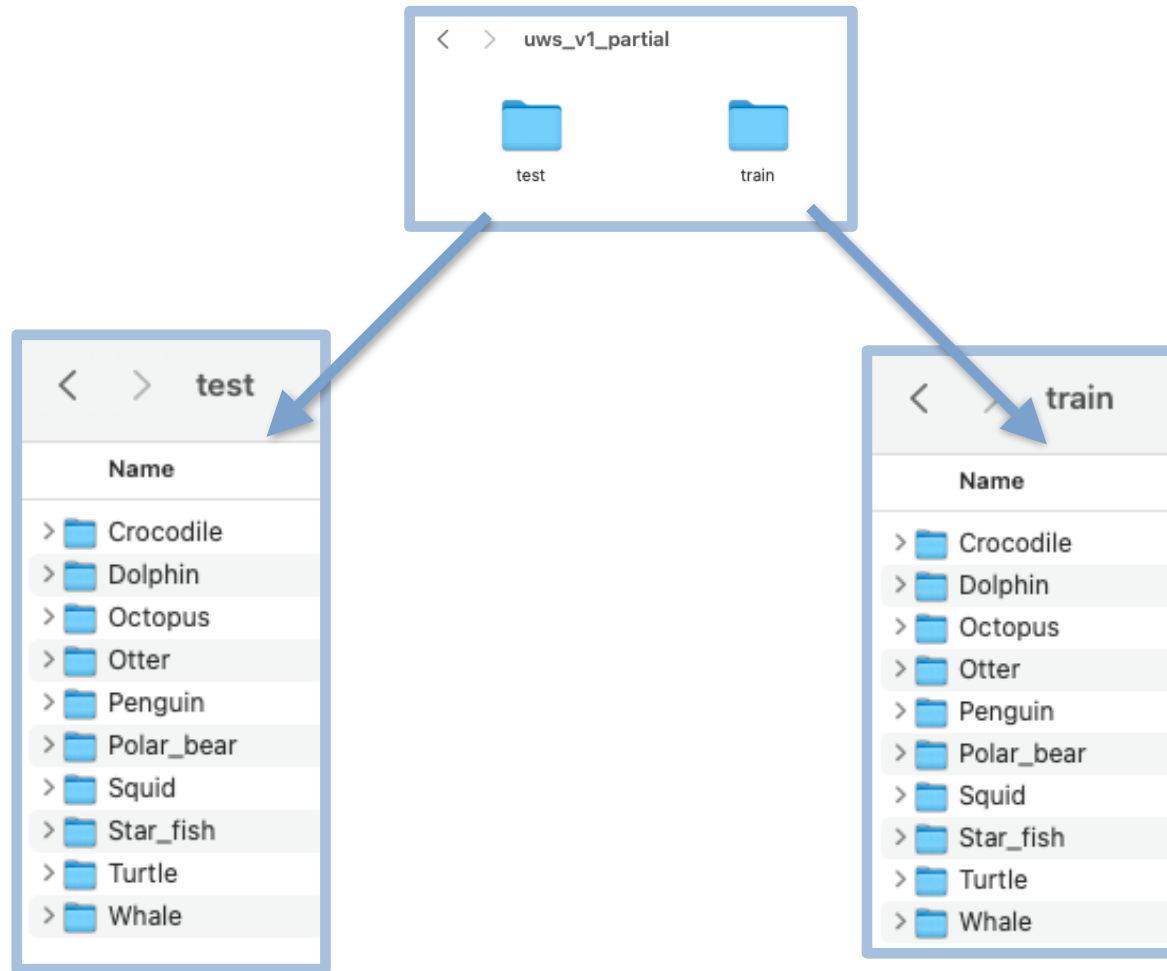


Image classification

<code>Caltech101(root[, target_type, transform, ...])</code>	Caltech 101 Dataset.
<code>Caltech256(root[, transform, ...])</code>	Caltech 256 Dataset.
<code>CelebA(root[, split, target_type, ...])</code>	Large-scale CelebFaces Attributes (CelebA) Dataset Dataset.
<code>CIFAR10(root[, train, transform, ...])</code>	CIFAR10 Dataset.

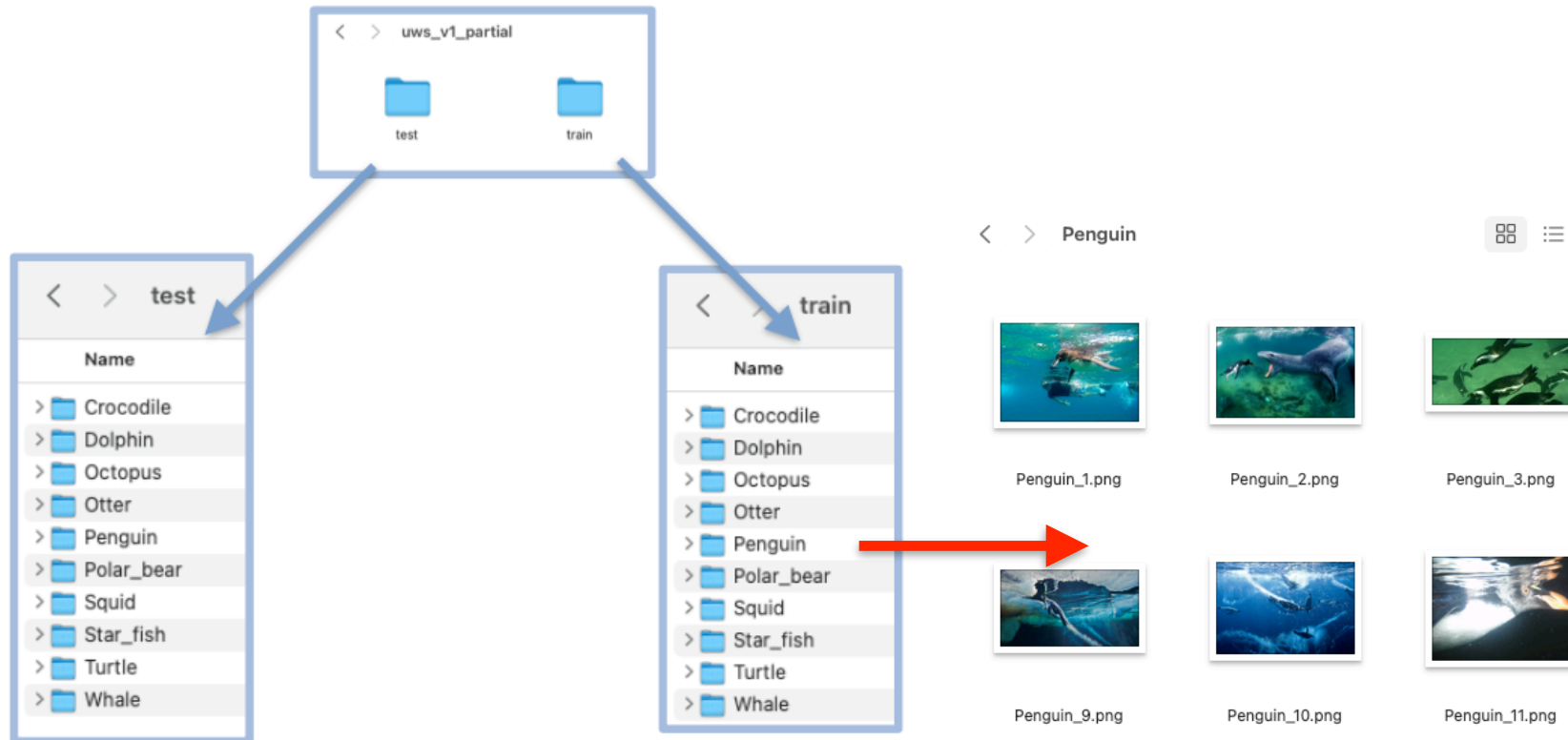
Fine-tuning AlexNet on an Arbitrary Dataset

- This dataset is organized into 'train' and 'test' folders as follows:



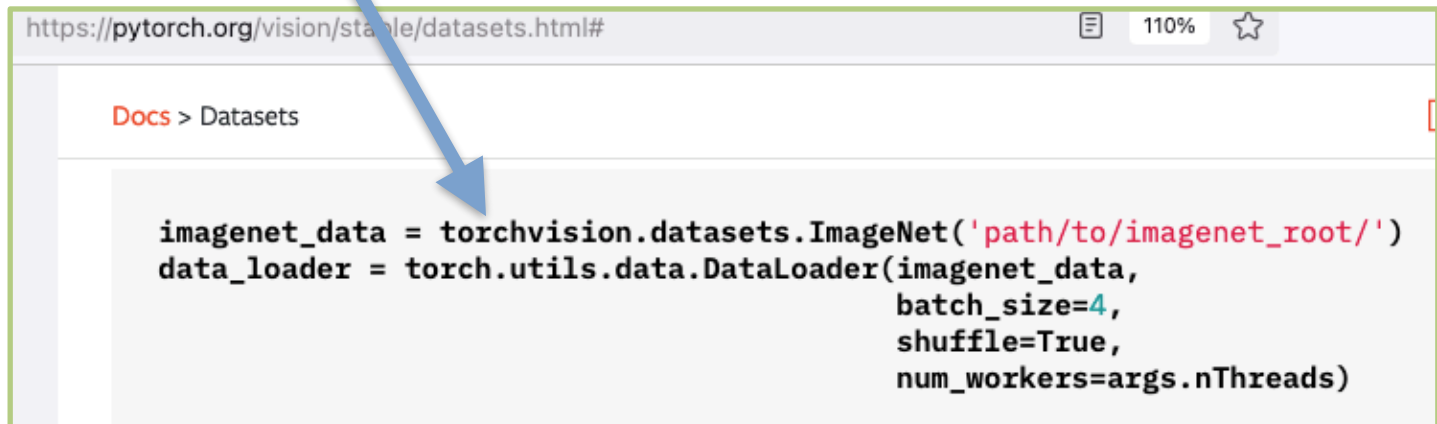
Fine-tuning AlexNet on an Arbitrary Dataset

- Each folder ('train' and 'test') contains a set of images that will be used by our model during fine-tuning and testing, respectively



Existing Dataset in PyTorch

- If we need to use PyTorch's existing datasets, we can use the following module from PyTorch to easily download and prepare the data loader for training and testing.

A screenshot of a web browser showing the PyTorch documentation page for datasets. The address bar shows the URL 'https://pytorch.org/vision/stable/datasets.html#'. The page content includes a breadcrumb 'Docs > Datasets' and a code block. A blue arrow points from the text in the first bullet point to the code block. The code block contains the following Python code:

```
imagenet_data = torchvision.datasets.ImageNet('path/to/imagenet_root/')
data_loader = torch.utils.data.DataLoader(imagenet_data,
                                           batch_size=4,
                                           shuffle=True,
                                           num_workers=args.nThreads)
```

- This is what we used in our previous experiment when training our own CNN from scratch using the CIFAR-10 dataset or Fashion-MNIST dataset.

Using Arbitrary Dataset

- Instead, when we need to use an arbitrary dataset, we can use the following module from PyTorch to prepare the data loader for training and testing.

```
TEST_IMAGE_SIZE_W = 227
TEST_IMAGE_SIZE_H = 227
mean, std = get_imagenet_mean_std_normalized()
print(f"ImageNet: mean: {mean}, std: {std}")

# CNN architectures such as AlexNet, VGGNet, and ResNet has been pre-trained using the ImageNet dataset.
# You need to normalize each image with the given mean and standard deviation before doing the forward-pass on these networks.
transform = transforms.Compose([
    transforms.Resize((TEST_IMAGE_SIZE_W, TEST_IMAGE_SIZE_H)),
    transforms.ToTensor(),
    transforms.Normalize(mean=mean, std=std) # ImageNet: mean (R, G, B) and standard deviation (R, G, B)
])

train_dir = '/content/drive/MyDrive/cs195_fall24/classification/datasets/uws_v1_partial/train'
test_dir = '/content/drive/MyDrive/cs195_fall24/classification/datasets/uws_v1_partial/test'

train_dataset = datasets.ImageFolder(train_dir, transform=transform)
test_dataset = datasets.ImageFolder(test_dir, transform=transform)

N_train = len(train_dataset)
N_test = len(test_dataset)

number_of_classes = 10
print("Number of classes: ", number_of_classes)
print("Size of train set:", N_train)
print("Size of test set:", N_test)
```

```
train_dataloader = DataLoader(train_dataset, batch_size=batch_size_val, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size_val, shuffle=False)
```

Loading a Pre-trained AlexNet Model in PyTorch

- Import a pre-trained instance of AlexNet inside our Network class and make any other necessary changes as follows:

```
class AlexNet(nn.Module):

    def __init__(self, num_classes, pretrained=True):

        super(AlexNet, self).__init__()

        # download PyTorch's own implementation of AlexNet model trained on ImageNet
        net = models.alexnet(pretrained=True)

        # retained weights for convolutional, pooling, linear layers from AlexNet
        self.features = net.features
        self.avgpool = net.avgpool
        self.classifier = net.classifier

        # IMPORTANT: "If you need to fine-tune this network for your own dataset
        # the simplest modification is to replace the last layer in self.classifier
        # the updated AlexNet has the desired number of output classes: 'num_classes'
        self.classifier[-1] = nn.Linear(4096, num_classes) # only this last layer

    def forward(self, x):

        print("shape of input: ", x.shape)
        x = self.features(x)
        print("output shape (self.features): ", x.shape)
        x = self.avgpool(x)
        print("output shape (self.avgpool): ", x.shape)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        print("output shape (self.classifier): ", x.shape)
        return x
```

Fine-tuning AlexNet on an Arbitrary Dataset

- Go to Blackboard and follow the notebook as shown below: